LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
I16r
no.439-444
cop.2

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

To renew call Telephone Center, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

DEC 23 1981

3-17-96

FEB 29 1996

L161—O-1096

Digitized by the Internet Archive
in 2013

http://archive.org/details/weedwonderfulequ444fraz

510.84
I86r
no 444

Report No. 444

Math

COO-1469-0181

WEED
A WONDERFUL EQUATION ELIMINATION DEVICE

by

CELSO JOHN FRAZAO GUIMARAES

June 1971

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 444

WEED
A WONDERFUL EQUATION ELIMINATION DEVICE*

by

CELSO JOHN FRAZAO GUIMARAES

June 1971

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

* This report was supported in part by the Department of Computer Science and in part by the Atomic Energy Commission under grant US AEC AT(11-1)1469, and submitted to the graduate college in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

E R R A T A T O

REPORT NO. 444 COO-1469-0181

WEED
A WONDERFUL EQUATION ELIMINATION DEVICE

by

CELSO JOHN FRAZAO GUIMARAES

June 1971

List of Corrections

p. 4, line 3

$I_1 = I_0 * \sin(* T)$ should be $I_1 = I_0 * \sin(\pi * T)$

p. 18, line 5

$E_1\ C = 0$ should be $E_1 * C = 0$

p. 18, line 6

$C\ E_1 = 0$ should be $C * E_1 = 0$

p. 18, line 7

$E_1\ \ E_2 = 0$ should be $E_1 ** E_2 = 0$

p. 18, line 8

$E_1 \pm E_2 = 0_3$ should be $E_1 \pm E_2 = 0$

p. 18, line 8

$-E_2$ should be $\pm E_2$

p. 18, line 9

$E_2\ E_3$ should be $E_2 * E_3$

p. 18, line 10

$E_2\ E_3$ should be $E_2 * E_3$

p. 18, line 11

$E_1 E_4 = E_2 E_3$ should be $E_1 * E_4 = E_2 * E_3$

p. 19, lines 15, 19, and 21

Change the 5.3's and 5.1 to 6.3's and 6.1.

p. 22, line 7

or '1' should be or '-1'

ACKNOWLEDGMENT

I am indebted to the following people whose assistance was most invaluable in the preparation of this thesis. To Professor C. W. Gear for suggesting the topic and for his enlightening guidance in times of panic. To Miss Barbara Hurdle for her unmatched patience and talent in typing the manuscript. To the Graphics Group (Al Whaley, Marty Michel, Roger Haskin, John Nickolls, and Sally Wilkins) for the fruitful "rap" sessions. And to the Department of Computer Science in general for so kindly providing all the necessary facilities.

PREFACE

This thesis is a description of WEED, an algorithm for simplifying systems of equations symbolically and classifying variables for further numerical analysis. It is an integral part of the Graphics-Oriented Simulation and Modeling System currently being developed at the University of Illinois.

Through a consistent method of elimination, WEED produces a condensed system of equations and an Equivalence Table for variables, which allow for a more efficient compilation and numerical integration than would otherwise be possible.

TABLE OF CONTENTS

Page

1. THE MODELING SYSTEM. 1

2. PRELIMINARIES AND OVERVIEW OF THE ALGORITHM. 3

3. INPUT SPECIFICATIONS 6

4. OUTPUT REQUIREMENTS. 10

5. EQUATION CLASSIFICATION. 14

5.1 Variable = Constant 16

5.2 $\underline{\text{Variable}_1 = \text{Variable}_2}$ 16

5.3 Variable = Expression 17

5.4 Expression = Expression 18

6. THE PROCESS ROUTINE. 19

6.1 Constant. 19

6.2 Global Variable or TIME 19

6.3 Other Variable. 19

6.4 Binary Operation. 20

6.5 Unary Operation 22

6.6 User-Defined Function 23

7. THE FINAL STAGES: NAME REASSIGNMENTS,

EQUATION RESTRUCTURING AND OUTPUT. 24

BIBLIOGRAPHY 26

APPENDICES 27

1. THE MODELING SYSTEM

The General Simulation and Modeling System currently being developed at the University of Illinois has as its primary objective "to provide a tool for the user with which he can (graphically) construct models of a proposed system and analyze them for static, dynamic and oscillatory behavior. The user will have the capability to define elements which are models of individual items of his system (e.g. diodes, motors, water pumps) and specify their behavior by means of differential and algebraic equations. The elements will have terminals (e.g. wires, mechanical links, pipes) through which certain variables may be transmitted (e.g. voltage, current, torque, displacement, pressure, flow). The terminals from several elements can be connected together at nodes (connection points) to form a network." A network may be used as an element of a larger network, and hence in a recursive fashion fairly complex networks can be constructed quite easily. The overall network is then the model of the proposed system, and is equivalent to the system of differential and algebraic equations derived from each element and node of the model. Thus the model can be completely analyzed by solving this equivalent system of equations.

The user interacts with the Modeling System through a drawing program which builds data structures representing each element and network. From these, Item and Global Analyses create new data structures which no longer contain graphical information. The equations are parsed into tree structures, assigned values are substituted in the equations, and all external names are replaced with new internal pointers. WEED then simplifies the equations, classifies them according to the types of operations and variables they contain, and weeds out those which are no

longer needed. Variables are organized in an Equivalence Table according to their original definition, and placed into different sets depending on how they are used in the equations. New internal names are then assigned to all the variables, and the equations are updated and restructured into a compact format. The final stage then compiles the condensed system of equations, and solves for transient, steady-state or oscillatory response.

2. PRELIMINARIES AND OVERVIEW OF THE ALGORITHM

Variables contained in the system of equations are not all of the same nature. Some refer to terminals in the network, and are identified as I-variables (e.g. electrical current). Others are associated with nodes and are classed as E-variables (e.g. voltage). Local variables (also called internal variables) are those which are specific to an instance of element in the network. Global variables, on the other hand, are common to groups of elements (e.g. ambient temperature). The special global variable TIME is permanently assigned. Finally, parameters are those variables associated with a given type of element, whose value is specified with each instance of that element in the network (e.g. resistance of a resistor).

In addition to keeping track of the above categories, WEED places all variables into four mutually exclusive sets, according to their usage in the equations:

a. Set S_1--all global variables and variables defined by S_1-equations (of the form V = function (globals and constants only)).

b. Set S_2--variables defined by S_2-equations (of the form V = function (globals, constants, and time)), and the special variable TIME.

c. Set L--variables involved only in linear expressions.

d. Set M--variables involved in non-linear or differential expressions. As an example, given the equations

$$E_0 = 3*E_1 + G_0*(E_2+E_3) - 1/E_4 + (E_5+2)*T - E_6*E_7 + E_8'$$

$$I_0 = 3 - G_0/2$$

$$I_1 = I_0*\sin(*T)$$

$$I_2 = (I_1+3)/(I_1-I_0)$$

where the E_i are E-variables, the I_i are I-variables, the G_i are globals and T is time, E_0 through E_3 would belong in set L; E_4 through E_8 in set M; I_0 in set S_1; and I_1 and I_2 in set S_2. Note that since global values are fixed throughout any given execution of the Modeling System, global variables are treated as constants insofar as variable and expression classification is concerned.

An additional set DY of new variables is created for the output of general equations, and is discussed in section 4.

A number of passes are made through the system of equations, each time advantage being taken of simplifications made in previous passes. As each equation is processed, all expressions which are only functions of constants are evaluated immediately and replaced by the result. Expressions of the form

$E\pm 0$, $0\pm E$, E*0, 0*/E, E*/1, 1*E, E**0, E**1,

1**E, and $\frac{d}{dt}$ (S_1 or S_2-expression)

are simplified as they are detected, and unary minuses are eliminated wherever possible. Equations which become trivial relations, such as

variable = constant

or

variable = $\pm$ variable

are eliminated by setting appropriate pointers in the Equivalence Table (described in section 4), and the remaining equations are placed

into three sets: S_1-equations, S_2-equations, and general equations (those which are neither S_1 nor S_2).

The reason for arranging all the equations and variables into different sets is that each set can receive special treatment in the numerical analysis phase, thereby increasing performance and reducing cost. For instance, variables in set S_1 need only be evaluated once per simulation; those in set S_2 once per time step; and derivatives of variables in set L need not be evaluated at all.

The equations are reprocessed until there are no more changes in the Equivalence Table (up to a maximum of three passes). All variables are then renumbered so that the members of each set are contiguous in memory--that is, the numbers 0 to $n_{S_1}-1$ refer to the n_{S_1} members of set S_1; n_{S_1} to $n_{S_1} + n_{S_2}-1$ to the n_{S_2} members of set S_2; $n_{S_1} + n_{S_2}$ to $n_{S_1} + n_{S_2} + n_L-1$ to the n_L members of set L; and $n_{S_1} + n_{S_2} + n_L$ to $n_{S_1} + n_{S_2} + n_L + n_M-1$ to the n_M members of set M. The new names are substituted in the equations, and these in turn are compressed and output in their three different sets to the numerical integration routines. A concise flow chart of the WEED algorithm is given in Appendix A.

3. INPUT SPECIFICATIONS

The input to WEED consists of a set of Input Tables and a single list of equations, as shown in Figure 1.

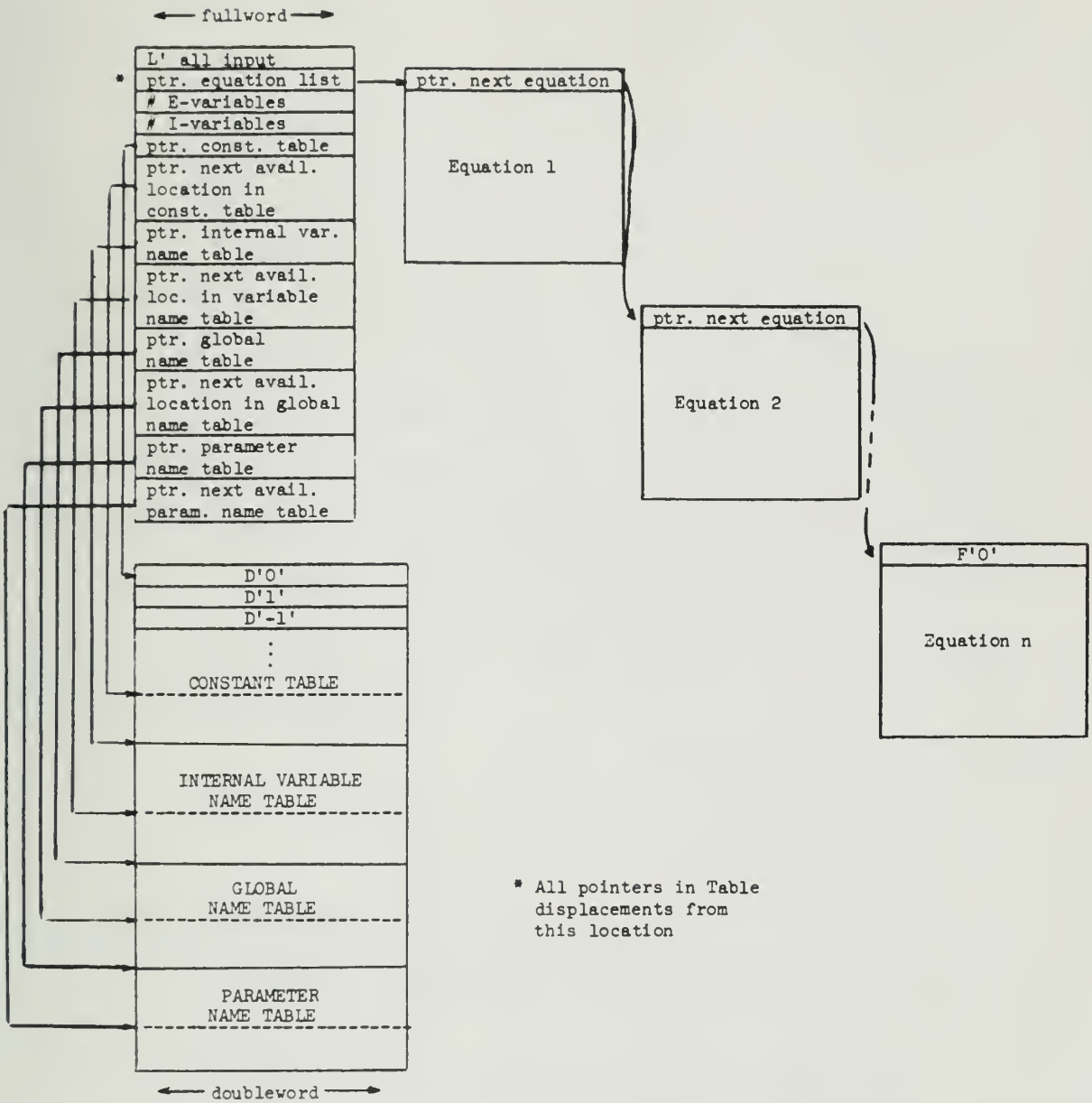


Figure 1. Input to WEED

The Name Tables contain the actual eight-character variable names provided by the user. In the equations, however, these variables are represented by the displacement of their name from the beginning of their respective Name Tables.[1] The same holds true for constants, where the Constant Table contains the double-precision floating-point constant values.

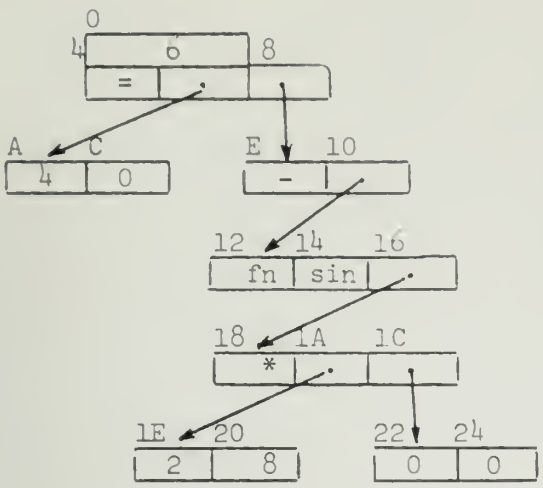
The equations are in tree-structured form, and consist of a series of "nodes" of information connected by pointers, as illustrated in Figure 2. The first node of each equation consists of a pointer to the next equation in the list, as a displacement from the current equation. The remaining nodes are clusters of halfwords (usually two or three)[2] contiguous in memory. The first HW of each node is an op-code which indicates the type of node. If the node represents an operation, the remaining HW-fields contain pointers to its respective operands.[3] If the node is a variable or constant, the second HW contains the internal name associated with it. Table 1 describes the different op-codes and associated operands.

---

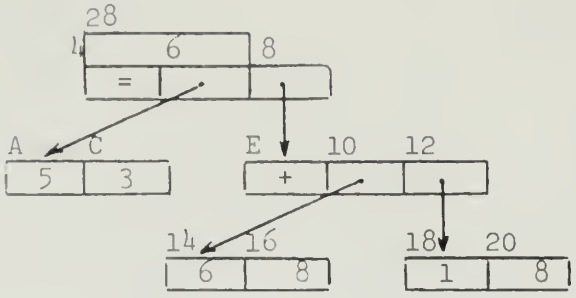
[1] E- and I-type variables do not have alphameric names associated with them, and are numbered sequentially, starting from 0.

[2] The 'user-specified function' node is a variable-length node, including fields for function type (second HW) and number of pointers (third HW).

[3] A 'function' node contains a field specifying the type of function (second HW).



$E_0 = -\sin(P_1 * T)$



$I_3 = 1 + G_1$

address	in memory
0	0000 0028
4	0007 000A 000E
A	0004 0000
E	0011 0012
12	0012 0006 0018
18	000A 001E 0022
1E	0002 0008
22	0000 0000
28	0000 0024
2C	0007 000A 000E
32	0005 0003
36	0008 0014 0018
3C	0006 0008
40	0001 0008

Figure 2. Equation Format

Node Type	Op	Operands	Halfwords
Time	X'0'	X'0'	2
Global Variable	1	Displ. in Global Name Table	2
Parameter	2	Displ. in Parm Name Table	2
Internal Variable	3	Displ. in Int. Var. Name Table	2
E-Variable	4	Numeric Name	2
I-Variable	5	Numeric Name	2
Constant	6	Displ. in Constant Table	2
Equals (=)	7	Left-pointer; Right-pointer	3
Addition (+)	8	L-ptr; R-ptr	3
Subtraction (-)	9	L-ptr; R-ptr	3
Multiplication (*)	A	L-ptr; R-ptr	3
Division (/)	B	L-ptr; R-ptr	3
Assignment (←)	C	L-ptr; R-ptr	3
Power (**)	D	L-ptr; R-ptr	3
Max	E	L-ptr; R-ptr	3
Min	F	L-ptr; R-ptr	3
Differentiation	10	Operand-ptr	2
Unary	11	Operand-ptr	2
Function	12	Fn ID#; Operand-ptr	3
User-spec fn	13	Fn ID#; #-operands;operand-ptrs	$\geq$ 3

Table 1. Internal Codes for Equation Nodes

4. OUTPUT REQUIREMENTS

The output consists of a Table Block and an Equation Block, as depicted in Figure 3. The Table Block consists of the Constant Table (including possible additions created in the WEED step), the input Name Tables (for use in cross-referencing and diagnostics), and the Equivalence Table (EQV), which contains all the variables ordered according to network type, with their respective new internal names (variables found to be equivalent will, of course, have the same assigned name). The Table Block also contains other necessary data, such as pointers to the various tables and to the Equation Block.

Upon entry to WEED, the variables are arranged in the EQV in order of ascending numeric names for each field in the table, as shown in Figure 4. As the equations are processed, different types are assigned to the variables, and appropriate pointers are set for those which are found to be equivalent to other variables or to constants. During internal name reassignment, all variables are marked 'assigned' (except for those equivalent to constants), so that upon output, only three types will be present in the EQV: assigned (+V), assigned (-V), and constants.

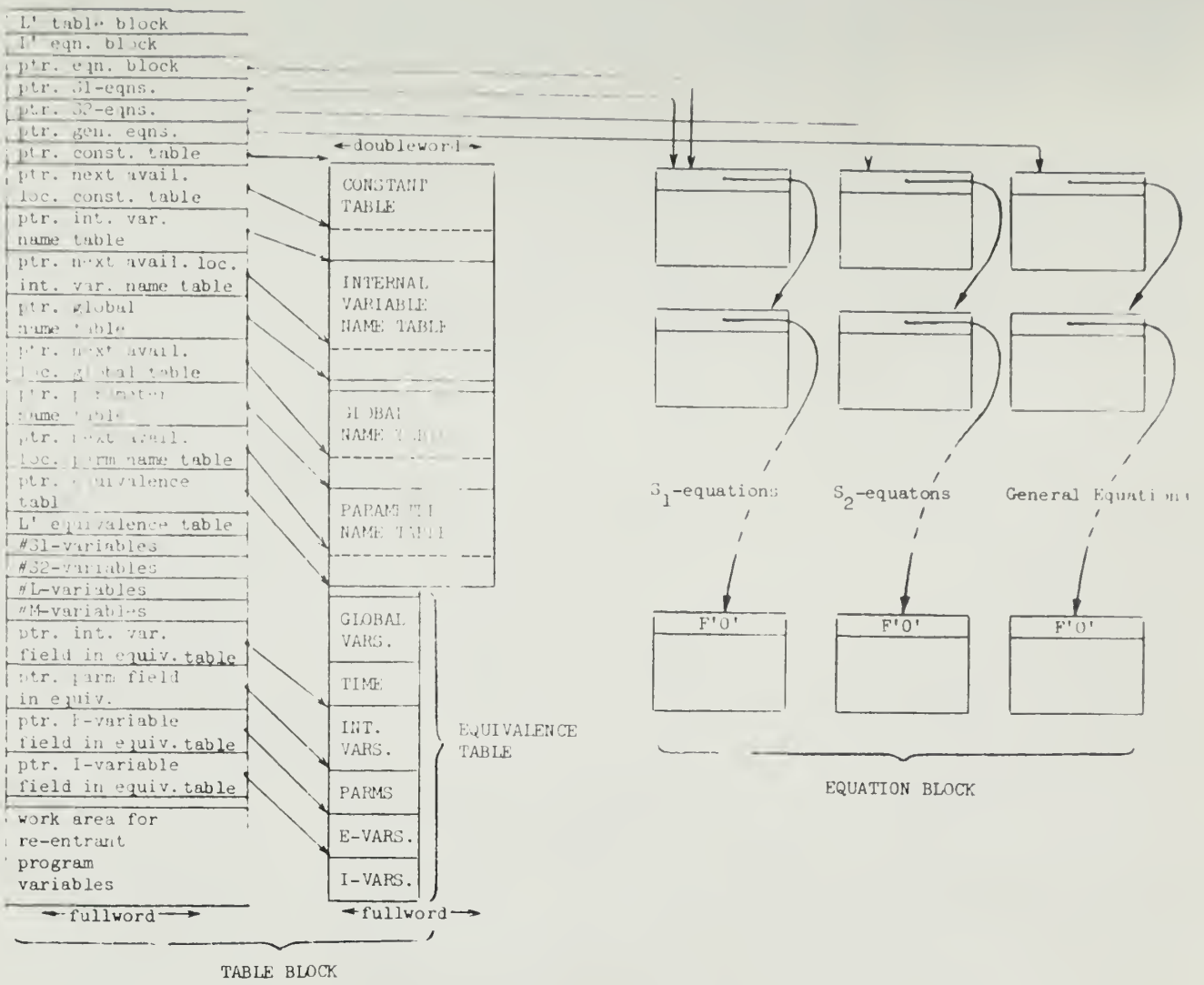
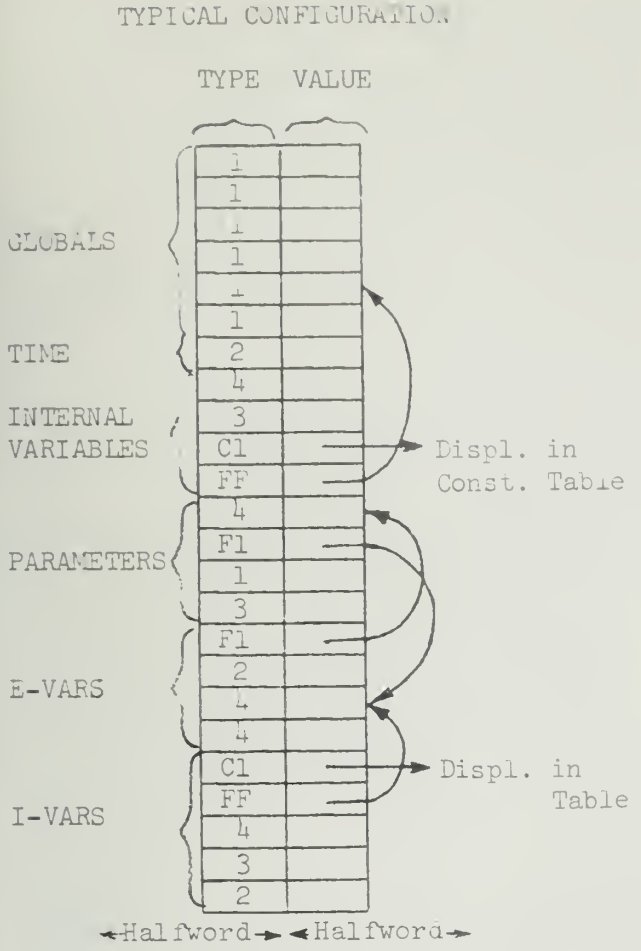


Figure 3. Output From UTED



EQV TYPE	REPRESENTATION 1st HW	2nd HW
Unused	X'00'	x'00'
[illegible]	01	00
[illegible]	02	00
L	03	00
M	04	00
Assigned (+V)	A1	new name
Assigned (-V)	AF	new name
Constant	C1	pointer to const. table
+Variable	F1	ptr. to var in EQV
-Variable	FF	ptr. to var in EQV

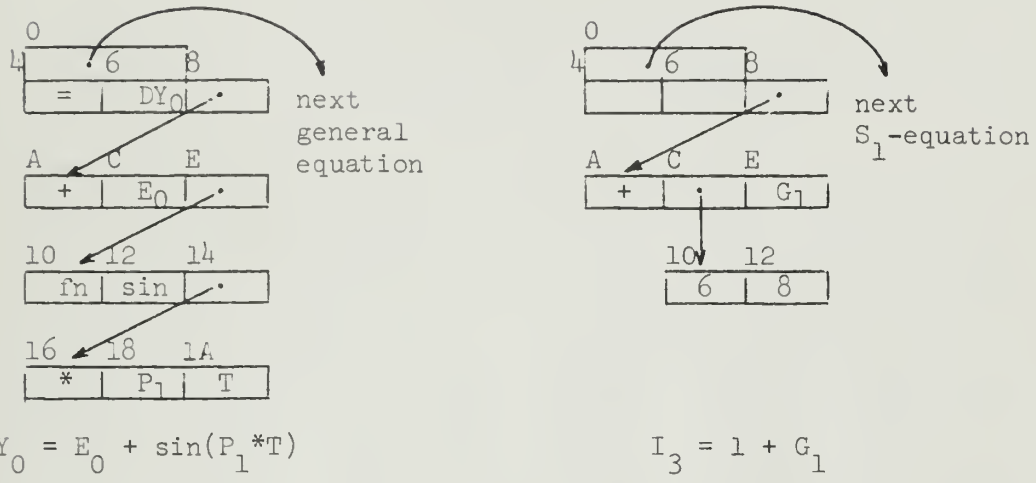
Figure 4. Equivalence Table

The Equation Block consists of three sub-blocks corresponding to the three different sets of equations. The equations take a form similar to the input equations, with two differences:

1. Operation nodes incorporate operands which are variables, tneir pointer fields being replaced by the corresponding new variable names. The remaining pointers to other nodes take on a negative value, in order tc differentiate from variable names.

2. General equations are put in the form 0 = expression, and the 0 is replaced by a new variable of the set DY, where the DY's are numbered starting at $n_{S_1} + n_{S_2} + n_L + n_M$.

Figure 5 illustrates the modifications performed on the example of Figure 2.



$DY_0 = E_0 + \sin(P_1 * T)$

$I_3 = 1 + G_1$

Address	In Memory
0	0000 001C
4	0007 0123 FFF6
A	0008 0069 FFF0
10	0012 0006 FFEA
16	000A 0033 0017

ASSUMING

var	new name
DY_0	123
E_0	69
P_1	33
T	17
I_3	269
G_1	1

Address	In Memory
0	0000 0014
4	0007 0269 FFF6
A	0008 FFF0 0001
10	0006 0008

Figure 5. Output Equations

5. EQUATION CLASSIFICATION

For this and the following sections, the flow charts provided in the Appendices should be useful as an aid to understanding the WEED Algorithm.

Upon entry, WEED goes through an initialization phase, in which it GETMAIN's an area for the output Table Block, copies the Name Tables and Constant Table from the input area, sets pointers to the different tables, and zeroes out the Equivalence Table. It is then ready for the Equation Classification stage.

WEED takes each equation in turn, replaces the '=' in the first node by a '-', and calls upon the PROCESS Routine (described in section 6) to determine the types of expressions on the left and righthand sides of the equation. PROCESS travels recursively down the tree, performing simplifications as it goes along, and returns two pairs of numbers representing the left and right expression types. Every expression has two numbers associated with it: one (referred to in the program as EXPTP, LX or RX) which gives an indication of the structure and complexity of the expression; and another (referred to as S1S2EXP, S1S2LX, or S1S2RX) which indicates whether the expression is type S_1, S_2, or neither. Table 2 lists the possible types and respective codes.

Expression Type (EXPTP, LX, RX)		Concurrently (S1S2EXP, S1S2LX, S1S2RX)	
0	constant '0' (S1S2EXP=1)	0	Neither S_1 nor S_2-expression
1	other constant (S1S2EXP=1)	1	S_1-expression (single consts. included)
2	single variable	2	S_2-expression
3	constant $\pm$ variable		
4	variable $\pm$ variable		
5	general expression		

Table 2. Expression Types

Once WEED receives the left and right expression types from PROCESS, it restores the '=' and tries to put the equation in a standard format, according to the following priorities:

1. S_1 or S_2-expressions on the right side. (If the left side is also an S_1 or S_2-expression an error is indicated, the equation is eliminated and the next equation is fetched.)
2. Single variable on the left side.
3. Doublet (C$\pm$V or $\pm$V $\pm$V) on the left side.

After the appropriate switches are made, if a doublet (or any expression of the form E$\pm$E) is found on the left, it is split up, the left branch of the doublet being transposed to the right side of the equation.[1] At this stage, the equation will necessarily be in one of the following forms:

---

[1] Any constants in a doublet are always kept on the left. Hence in a split-up, the constant always gets moved to the right side of the equation.

variable = constant

variable = ± variable

variable = expression

expression = expression

We can now describe the individual steps taken in each case.

5.1 Variable = Constant

Set variable type to 'constant' (X'C1') in the EQV. Place a pointer to the constant (as a displacement from the beginning of the Constant Table) in the EQV value field of the variable. Eliminate the equation, and set EQVFLAG to 1, indicating a change made to the EQV.

5.2 Variable$_1$ = Variable$_2$

The two variable types determine the action to be taken, according to the following table:

Variable$_1$ \ Variable$_2$	S_1	S_2	L	M	C	+V	-V
S_1	E	E	↓↓	↓↓	E	↓↓	↓↓
S_2	E	E	↓↓	↓↓	E	↓↓	↓↓
L	OK	OK	OK	OK	OK	OK	OK
M	OK	OK	↓↓	OK	OK	OK	OK
C	E	E	↓↓	↓↓	E	↓↓	↓↓
+V	↷	↷	↓↓	↓↓	↷	↷	↷
-V	↷	↷	↓↓	↓↓	↷	↷	↷

E: Error. Relay message to the user and delete the equation.

⇌: Switch the variables, so that variable$_1$ becomes variable$_2$ and vice versa. Refer to the table.

↷: Follow the pointer (or chain of pointers)[2] beginning at variable$_1$. The last variable in the chain becomes the new variable$_1$. Return to the table.

OK: Set the type of variable$_1$ to +V or -V according to the equation which defines it. If variable$_2$ points to another variable, follow the pointer chain.[2] Place a pointer to the new variable$_2$ in the value field of variable$_1$.[3] Eliminate the equation and set EQVFLAG to 1.

5.3 Variable = Expression

If the expression is neither a S_1- nor S_2-expression, go to section 5.4 (Expression = Expression). Set the variable type to S_1 or S_2 accordingly. If the equation has already been placed in the proper equation set, proceed to the next equation. Otherwise, mark the equation by placing a 1 or 2 in the first byte of its link field, and set EQVFLAG to 1.

---

[2] In order to avoid following long chains in future searches through the EQV, a direct pointer is set from the variable being traced to the last variable in the chain.

[3] In the case where variable$_2$ points to a constant, variable$_1$ is made to point to the same constant (or its complement) and its type is set to 'constant.'

5.4 Expression = Expression

If the equation is in any of the following forms, simplify it, retest it for further simplifications and reprocess it:

$E_1/E_2 = 0 \rightarrow E_1 = 0$

$E_1 C = 0 \rightarrow E_1 = 0$ for $C \neq 0$

$C E_1 = 0 \rightarrow E_1 = 0$ for $C \neq 0$

$E_1 E_2 = 0 \rightarrow E_1 = 0$

$E_1 \pm E_2 = 0 \rightarrow E_1 = -E_2$

$E_1/E_2 = E_3 \rightarrow E_1 = E_2 E_3$

$E_3 = E_1/E_2 \rightarrow E_1 = E_2 E_3$

$E_1/E_2 = E_3/E_4 \rightarrow E_1 E_4 = E_2 E_3$

If from the first test it is found that the equation fits none of the above categories, place it in the set of general equations (by putting a 3 in the first byte of its link field) and proceed to the next equation.

6. THE PROCESS ROUTINE

Given a pointer to the top node of an expression, PROCESS will simplify the expression and determine its type. The types of lower level expressions (i.e. operand expressions) are used in determining the type of the current expression. Thus pairs of numbers (EXPTP,S1S2EXP) representing each lower level expression are transmitted recursively, from the bottom up.

The top node of each expression determines the subsequent steps to be taken, as follows:

6.1 Constant

Determine whether the constant is zero or non-zero, and set EXPTP to 0 or 1 accordingly. Set S1S2EXP to 1. Return.

6.2 Global Variable or TIME

Set EQV type to S_1 if node is a global, or to S_2 if it is TIME. Go to 5.3.

6.3 Other Variable

If unused so far, make the variable type L. If type L or M, set S1S2EXP to 0. If type S_1 or S_2 set S1S2EXP to 1 or 2, respectively. If type +V or -V, follow pointer chain and repeat 5.3. If type 'constant,' replace node in tree with a corresponding constant node, and go to 5.1. Set EXPTP to 2 (single variable) and return.

6.4 Binary Operation

Process the left and right expressions, remembering their respective types. S1S2EXP for the current expression is determined from the following table:

S1S2RX (right)

S1S2LX (left) \ S1S2EXP	0	1	2
0	0	0	0
1	0	1	2
2	0	2	2

Simplify unary minuses according to the rules of algebra, placing any remaining unary minus above the current node (in most cases, unary minuses can be successively bubbled up in this manner, so that upon reaching the top of the equation at most one unary minus remains).

If the operation is addition or subtraction, eliminate any zeros[1] on either side of the expression. Add or subtract constants appearing in constant nodes or doublet expressions (C±V), keeping the resultant constant on the left branch (either of the current node or of the left doublet). Determine EXPTP from the left and right expression types according to the table below, and return.

---

[1] If the top node of the equation is being processed (with '=' replaced by '-') zeros are preserved.

RX

LX \ EXPTP (+/-)	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	1	3	3	5	5
2	2	3	4	5	5	5
3	3	3	5	5	5	5
4	4	5	5	5	5	5
5	5	5	5	5	5	5

If the operation is multiplication or division, eliminate any 1's or -1's on either side of the expression. If either side is 0, make the current node a 0.[2] If both sides are constants, evaluate the expression. From S1S2LX and S1S2RX determine whether the expression is non-linear under multiplication or division, as follows:

S1S2RX

S1S2LX \ CALL NONLIN	0	0	2
0	*,/		*,/
1	/		
2	*,/		

If it is non-linear, call NONLIN (a routine which sets all L-type variables in the subtree to M-type and marks the top node of the expression as non-linear by making its op-code negative. This way, if a higher level expression is also found to be non-linear, the present subtree will not have to be rescanned). Finally, determine EXPTP from the table below, and return.

---

[2] Division by zero causes an error message and automatic elimination of the equation.

RX

LX: EXPTP (*,/)		0	1		2	3	4	5
			'1' or '-1'	other				
0		0	0	0	0	0	0	0
1	'1' or '1'	0	1	1	2	3	4	5
	other	0	1	1	5	5	5	5
2		0	2	5	5	5	5	5
3		0	3	5	5	5	5	5
4		0	4	5	5	5	5	5
5		0	5	5	5	5	5	5

For the case of a power operation, if both operands are constants, perform the exponentiation (C_1**C_2 evaluated as $e^{C_2 * \ln C_1}$), and set EXPTP to 0 or 1 depending on the result. Perform the following simplifications if possible:

E**0 → 1 (EXPTP ← 1)

E**1 → E (EXPTP ← LX)

0**E → 0 (EXPTP ← 0)

1**E → 1 (EXPTP ← 1)

If no simplification is possible, call NONLIN and set EXPTP to 5 (general expression). Return.

If the operation is max or min evaluate the expression if both operands are constants. Otherwise, call NONLIN and make EXPTP = 5. Return.

6.5 Unary Operation

Process the left operand. S1S2EXP is passed from operand to operator unchanged (i.e. S1S2EXP ← S1S2LX).

If the operation is differentiation, move any unary minus upwards. If the operand is an S_1-expression, replace the current expression with the constant '0' and set EXPTP to 0. Otherwise, call NONLIN and make EXPTP = 5. Return.

For a unary minus operation, if the operand starts with a unary minus, delete both unary minuses. If the operand is a constant, complement it and delete the unary minus. Copy EXPTP from the operand (i.e. EXPTP ← LX) and return.

If the operation is function, evaluate it if the operand is a constant, replacing the expression with the result, and setting EXPTP to 0 or 1. Otherwise, call NONLIN and set EXPTP to 5. Return.

6.6 User-Defined Function

Process all the operand subexpressions, call NONLIN, set S1S2EXP to 0 and EXPTP to 5. Return.

7. THE FINAL STAGES: NAME REASSIGNMENTS, EQUATION RESTRUCTURING AND OUTPUT

Having processed the entire system of equations, WEED counts the numbers of independent S_1, S_2, L and M variables in the EQV and initializes each of four registers with the first numeric name to be assigned to variables in each of the four sets. As the EQV is scanned, each time a new variable is encountered its value field is overwritten with the contents of its associated register, its type set to 'assigned(+V)' and the register incremented by 1. Whenever a variable type +V or -V is encountered, the new name of its equivalent variable is copied in its value field, and its type set to 'assigned(+V)' or 'assigned(-V)', accordingly. 'Constant' type variables remain unchanged. As a final result, all variables will have been numbered sequentially, with all S_1-variables first, S_2-variables next, and so on.

At this point, the equations are modified so that the new variable names are contained <u>within</u> operation nodes. As the equations are processed, a cumulative byte count is kept for each of the three equation sets, so that an appropriate amount of core can be later requested for the output Equation Block. Each tree is traversed recursively by means of the TRAVEL Routine (MODFORM mode). As each variable is detected, it is incorporated in its parent node by over-writing the respective pointer field. An exception if made for the case where the variable is 'assigned(-V)', in which case the old variable is transformed into a unary- node whose operand field contains the new variable name. Constants are not incorporated in parent nodes, and all remaining node-pointer values are made negative.

Once all the equations have been reformatted, WEED issues a GETMAIN for the output Equation Block and calls upon the PUTOUT Routine to supervise the output operation. PUTOUT looks only at S_1-equations to begin with, and passes them one by one to TRAVEL (TRANSFER mode), which recursively moves individual nodes to consecutive memory locations in the output area. Since the equations are squeezed into a compact form, the displacement relationships between the nodes are altered, and hence new pointer values must be set as the nodes are moved. New link pointers to the next equation can be set as soon as each equation transfer is completed.

PUTOUT repeats this process for S_2 and General Equations, and returns to WEED after setting pointers to the three sets of equations in the Table Block. Finally, the entire input area is FREEMAINed, the compiler is called, and WE ARE DONE.

BIBLIOGRAPHY

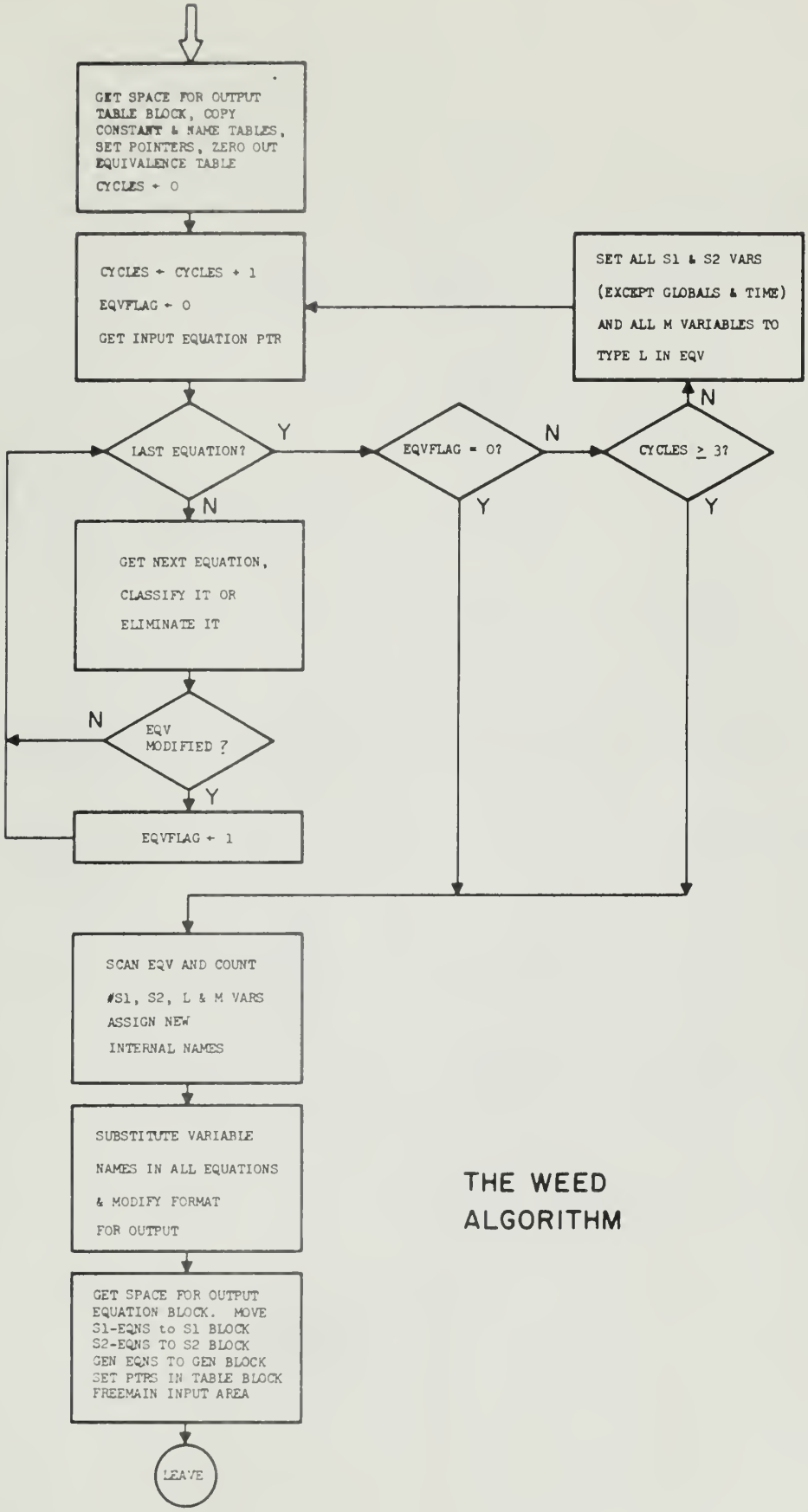
Gear, C. W., Hyde, C., Levin, H., Michel, M. J., Ratliff, K., and Wilkins, S., "The Simulation and Modeling System-- A Snapshot View," Department of Computer Science File No. 824, University of Illinois, Urbana, Illinois (1970).

"IBM System/360 Operating System: Assembler Language," IBM Systems Reference Library File No. S360-21 form GC28-6514-6.

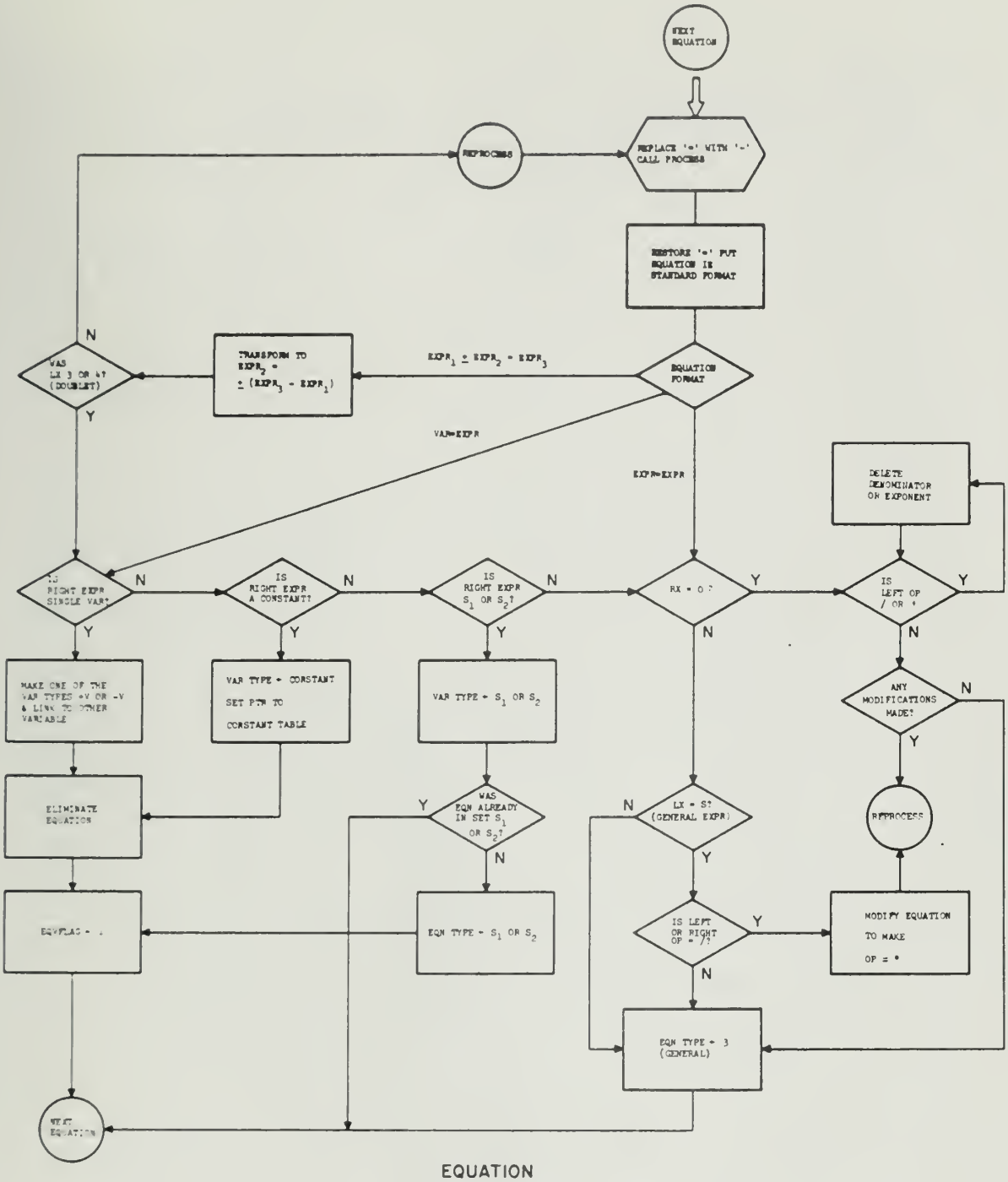
"IBM System/360 Operating System: Linkage Editor," IBM Systems Reference Library File No. S360-31 form C28-6538-6.

"IBM System/360 Operating System: Supervisor and Data Management Macro Instructions," IBM Systems Reference Library File No. S360-36 form GC28-6647-3.

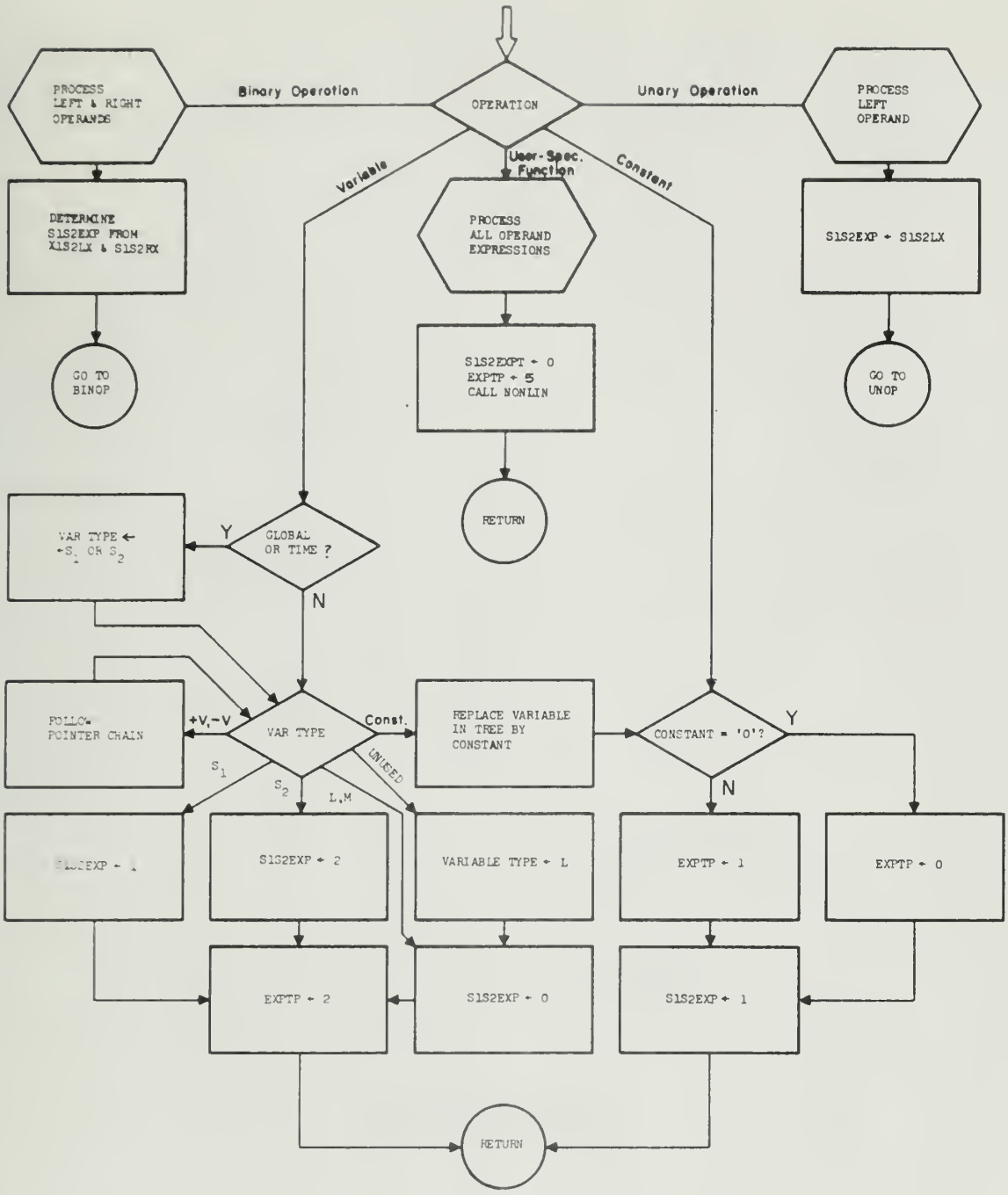
APPENDICES



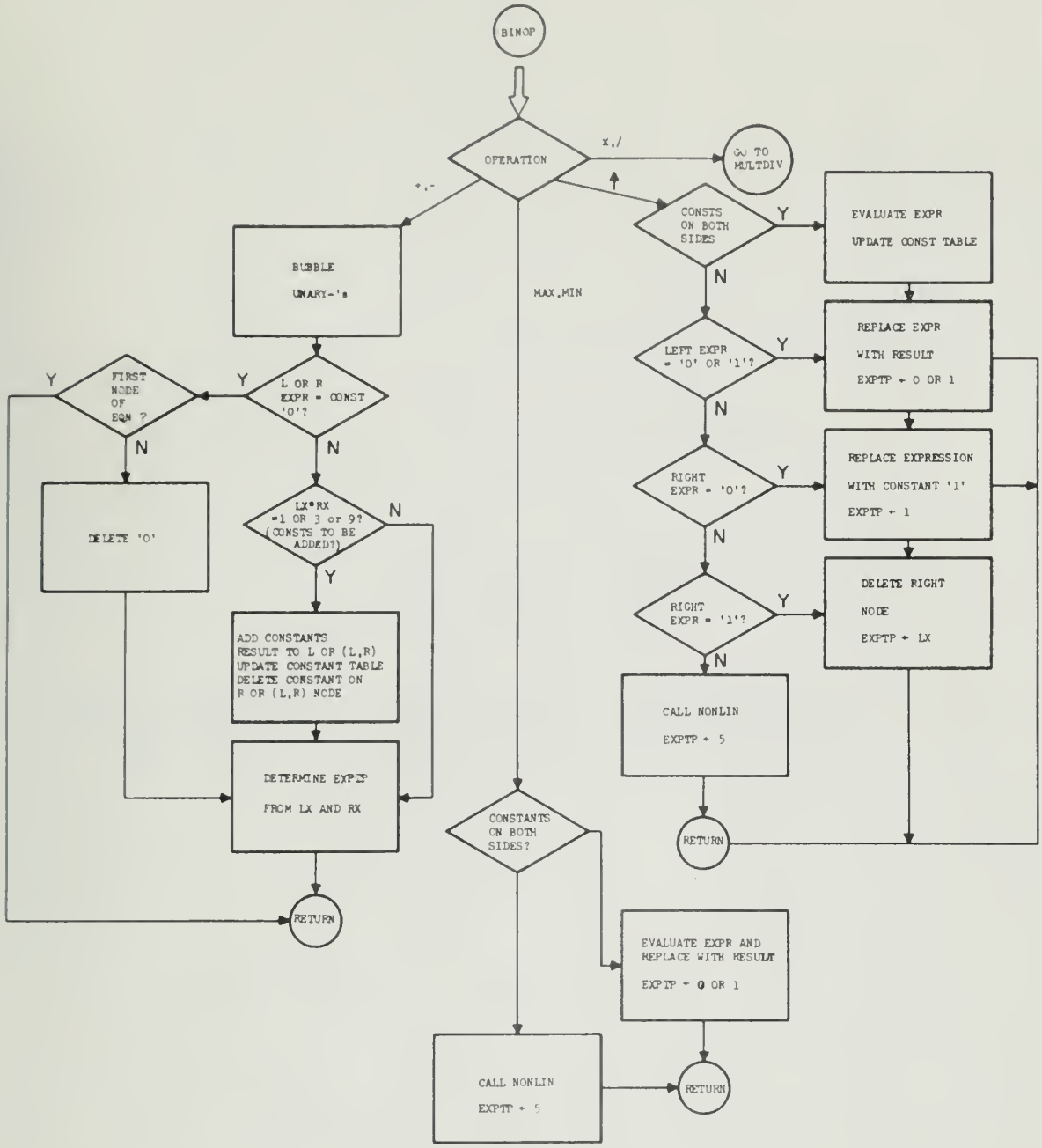
THE WEED ALGORITHM



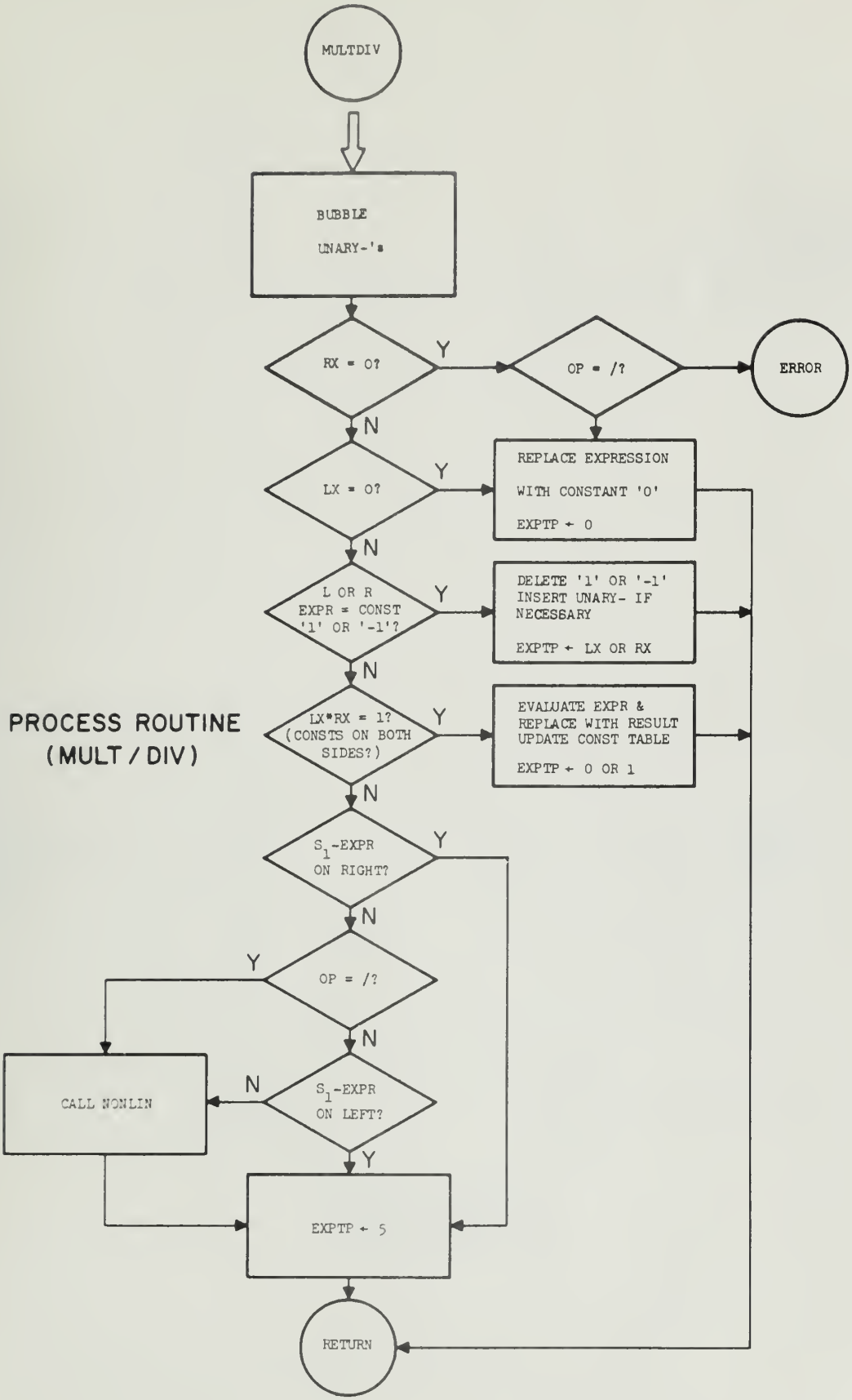
EQUATION
CLASSIFICATION



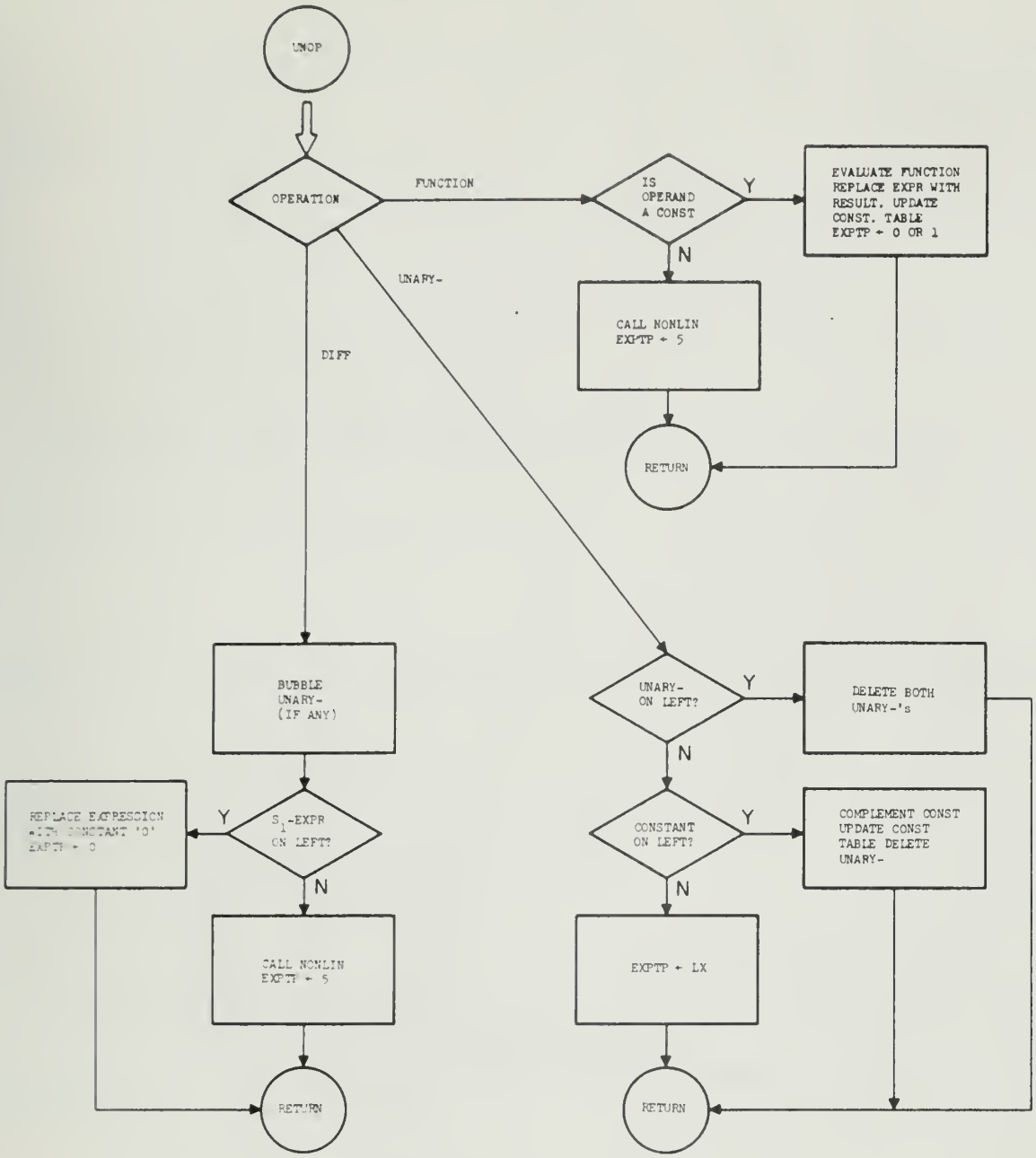
THE PROCESS ROUTINE



PROCESS ROUTINE
(BINARY OPERATIONS)



PROCESS ROUTINE
(MULT / DIV)



PROCESS ROUTINE
(UNARY OPERATIONS)

Form AEC-427
(5/68)
AECM 3201

U.S. ATOMIC ENERGY COMMISSION
UNIVERSITY–TYPE CONTRACTOR'S RECOMMENDATION FOR DISPOSITION OF SCIENTIFIC AND TECHNICAL DOCUMENT

(See Instructions on Reverse Side)

1 AEC REPORT NO.
COO-1469-0181

2 TITLE
WEED
A WONDERFUL EQUATION ELIMINATION DEVICE

3 TYPE OF DOCUMENT (Check one):

- ☐ a. Scientific and technical report
- ☐ b. Conference paper not to be published in a journal:
  - Title of conference _____
  - Date of conference _____
  - Exact location of conference _____
  - Sponsoring organization _____
- ☒ c. Other (Specify) Thesis

4 RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION (Check one):

- ☒ a. AEC's normal announcement and distribution procedures may be followed.
- ☐ b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
- ☐ c. Make no announcement or distrubution.

5 REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY NAME AND POSITION (Please print or type)

C. W. Gear
Professor and
Principal Investigator

Organization

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Signature Charles W. Gear

Date June 1971

FOR AEC USE ONLY

7 AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION

8 PATENT CLEARANCE:

- ☐ a. AEC patent clearance has been granted by responsible AEC patent group.
- ☐ b. Report has been sent to responsible AEC patent group for clearance.
- ☐ c. Patent clearance not required.

JUL 15 1971

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.436-444(1971
Interval generalization of switching the
3 0112 088399636